Exercise: 3

Name: Pekka Lehtola

How many tasks did you do: 9

Were the tasks easy, ok, difficult: The tasks were challenging this time, especially 3.7 produced problems.

Do you need help/comments in any task (if yes, to which ones):

No, but go easy with 3.7, I tried to do it for several hours and had to settle for an easier way because I didn't get it to work with another class at all.

1. Explain the following terms:

a. Abstraction (in programming)

- In object-oriented programming, abstraction is one of three central principles (along with encapsulation and inheritance). Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.
- What I read online about abstraction its like creating a private class that cant have objects, but with it other classes can be created.
- In the code ABC (Abstract Base Classes) is used to crate abstract class.

```
from abc import ABC, abstractmethod
class Payment(ABC):
  def print_slip(self, amount):
    print('Purchase of amount- ', amount)
  @abstractmethod
  def payment(self, amount):
    pass

class CreditCardPayment(Payment):
  def payment(self, amount):
    print('Credit card payment of- ', amount)

class MobileWalletPayment(Payment):
  def payment(self, amount):
    print('Mobile wallet payment of- ', amount)

obj = CreditCardPayment()
obj.payment(100)
obj.print_slip(100)
print(isinstance(obj, Payment))
obj = MobileWalletPayment()
obj.payment(200)
obj.print_slip(200)
print(isinstance(obj, Payment))
```

b. Accessor and mutator methods

- Accessor and mutator are basically get and set methods used in python.
- Accessor(get) returns copy of a private variable.
- Mutator(set) is used to modify private variables.

c. Public and private methods

- Private methods are basically the same as private attributes, they cant be called directly.
- They can be called with other classes methods or with the help of name mangling

```
# Python program to
# demonstrate private methods

# Creating a class
class A:

    # Declaring public method
    def fun(self):
        print("Public method")

    # Declaring private method
    def __fun(self):
        print("Private method")

    # Calling private method via
    # another method
    def Help(self):
        self.fun()
        self.__fun()

# Driver's code
obj = A()
obj.Help()
```

d. __str__ method (in Python)

- Used for defining output print of an object. With out it if you try to print an object, output would look like (<__main__.Test object at 0x00000286441D0A48>)

2. Modify the Coin class (see Exercise2) so that in addition to sideup you have another data attribute called currency. Add a function generating the currency (Euro, Pound, Dollar, Ruble, Yen). Use a random generator to get the currency (=similar to tossing the coin). Add a function to print out the currency.

3. Add a method that can change the currency of the coin. Test that your coin still works.

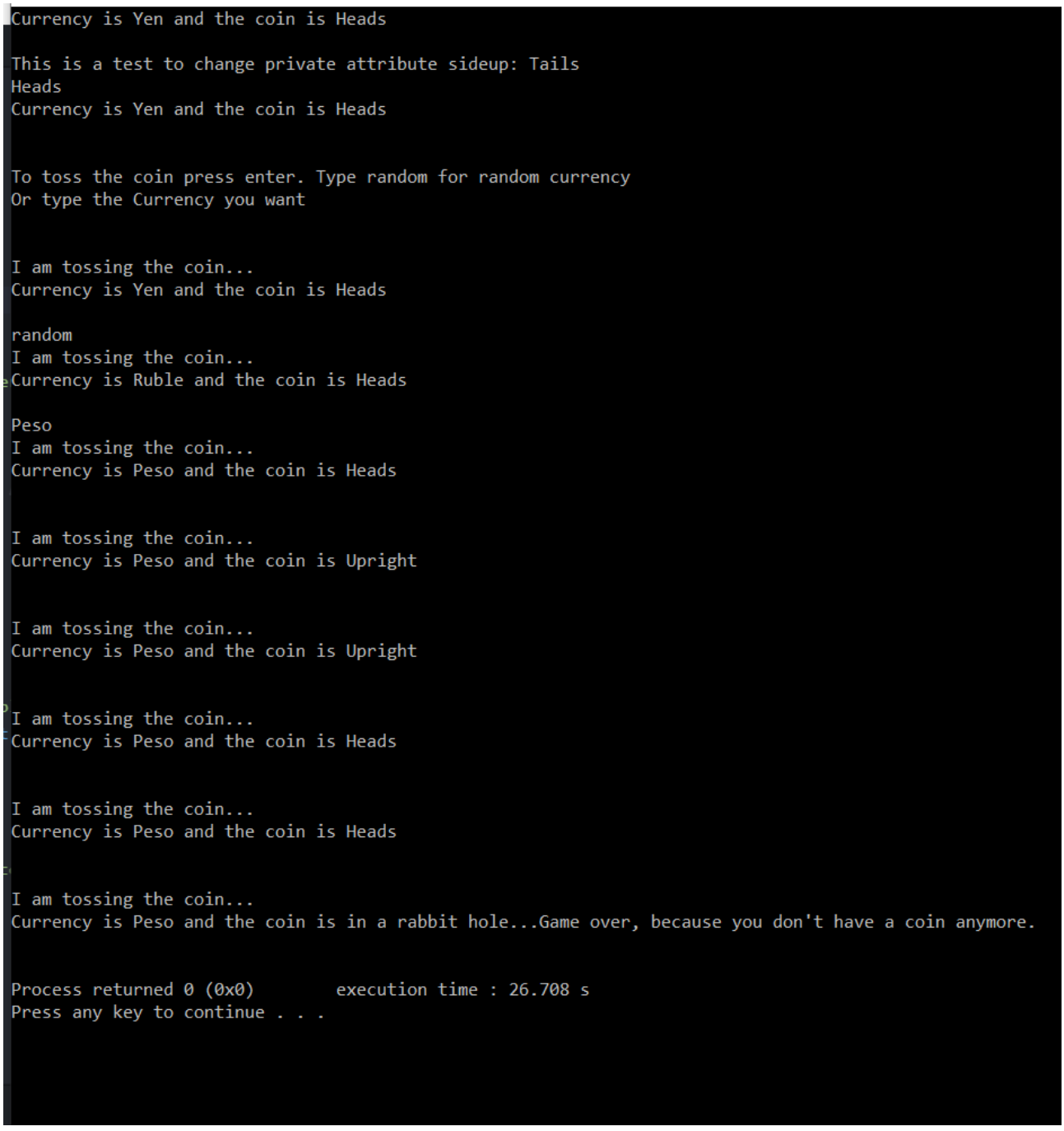
4. Change the Coin's sideup attribute to private. Test that your coin still works. What happens, if you now try to change the attribute's value from the main function? Try it out...!

Screen capture of Task 2/3/4

```
# File: Example
# Description: Simulate coin flip
# Author: Pekka Lehtola

import random

class Coin:
    # The __init__ method initializes the sideup data attribute with "heads"

    def __init__(self):
        self.__sideup = "Heads"
        self.currency = "Euro"



    # Selects a random integer between 0 and 14. If the value is 0-4 side up is heads.
    # If the value is 5-9 tails is choosen. 10-12 coin lands upright.
    # 13-14 coin falls into an rabbit hole...

    def toss(self):

        random_number = random.randint(0,14)

        if random_number in range(0,5):
            self.__sideup = "heads"

        elif random_number in range(5,10):
            self.__sideup = "tails"

        elif random_number in range(10, 13):
            self.__sideup = "upright"

        else:
            self.__sideup = "in a rabbit hole...Game over, because you don't have a coin anymore."
            print(self.__str__())
            exit()
```

```
36
37      # The get_sideup method return the value referenced by sideup.
38      def get_sideup(self):
39
40          return self.__sideup
41
42      # Returns a random currency from the list.
43      def get_currency(self):
44
45          currency_list = ["Euro", "Pound", "Dollar", "Ruble", "Yen"]
46          self.currency = str(currency_list[random.randint(0,4)])
47          return self.currency
48
49      # Str method for printing objects status.
50      def __str__(self):
51          return f"""Currency is {self.currency} and the coin is {self.get_sideup()}\n   """
52
53  # The main function
54  def main():
55
56      # Create an object from the Coin class.
57      my_coin = Coin()
58
59      # Get random currency
60      my_coin.get_currency()
61
62      # Display the side of the coin that is facing up.
63      print(my_coin)
64
65      # Atempting to change private attribute sideup in main
66      my_coin.__sideup= input("This is a test to change private attribute sideup: ")
67      print(my_coin.get_sideup())
68      print(my_coin, end="\n\n")
69
70      print("To toss the coin press enter. Type random for random currency")
71      print("Or type the Currency you want", end="\n\n")
72
73       while True:
74
75          # Makes the user press enter before next toss.
76          # If input is None currency stays the same.
77          # If user types random, new currency is chosen.
78          # If input is anythin else currency changes to that.
79          user_input = input()
80
81          if user_input !="":
82              if user_input =="random":
83                  my_coin.get_currency()
84              else:
85                  my_coin.currency = user_input
86
87          # Toss the coin.
88          print("I am tossing the coin...")
89          my_coin.toss()
90
91          # Display the side of the coin that is facing up.
92          print(my_coin)
93
94
95  # Call the main function.
96  main()
```

Screen capture of the output of Task **2/3/4**



Tämä oli yhdistelmä tehtävistä 2,3,4

Alussa yritys vaihtaa yksityisen atribuutin arvoksi Tails, joka ei toiminut.

5. Create a class Dice and make an object of it. You shall be able to roll the dice, get the result (number between 1 – 6) and get its color. Add at least 1 extra feature. Design your program using pseudocode. Document your code properly (with good comments) and pay attention to the clarity of the output prints.

Program starts

Initialize Dice:

Color = White

Side up = 6

Sum of throws = 0

Defining dice roll:

random number = random number between 1-6

color list = (1 = Red, 2 = Blue, 3 = Green, 4 = Yellow, 5 = Black, 6 = White)

Sum of throws = Sum of throws + random number

Dices side up = random number

Dices color = color list(random number)

Defining get side up:

print Dices side up

print sides color

print sum of throws

Defining restart game:

Initialize Dice again

Infinite Loop:

take user input

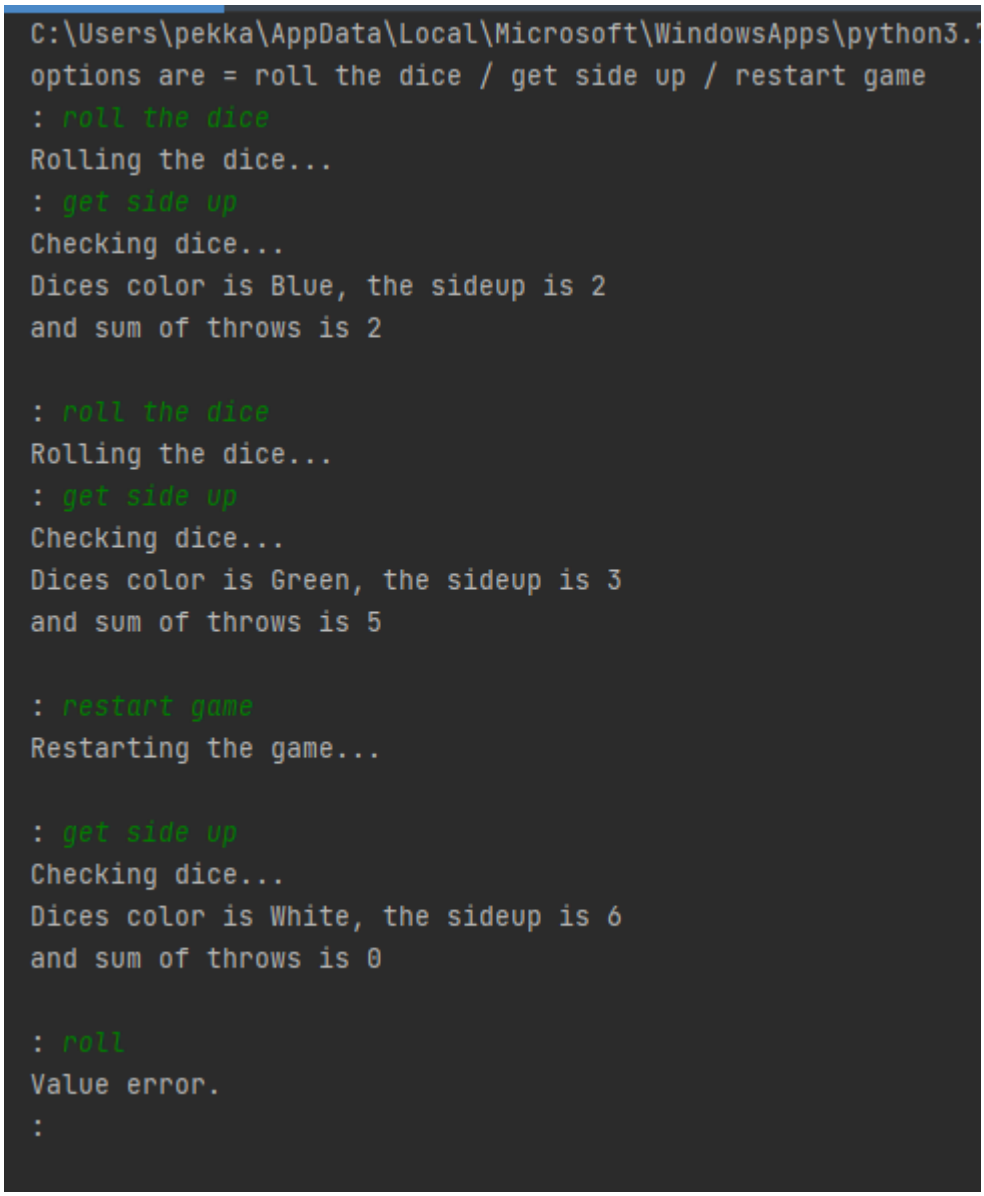
from that user input run:

dice roll , get side up or restart game

Screen capture of Task 5

```python
# File name: Exercise3_5
# Author: Pekka Lehtola
# Description: Roll a dice.

import random

class Dice:

    # Dice with three attributes: Color,
    def __init__(self):

        self.color = "White"
        self.side_up = 6
        self.sum_of_throws = 0

    # Selecting random number between 1 and 6, form that selectin predefined side color.
    # Modifying dices values and calculating new sum.
    def roll_the_dice(self):

        random_number = random.randint(1,6)
        colors = {1: "Red", 2: "Blue", 3: "Green", 4: "Yellow", 5: "Black", 6: "White"}

        self.sum_of_throws += random_number
        self.side_up = random_number
        self.color = colors[self.side_up]

        print("Rolling the dice...")

    # Prints objects str method.
    def get_side_up(self):

        print("Checking dice...")
        print(self)

    # Runs init method again to reset the sum of throws
    def restart_game(self):

        self.__init__()
        print("Restarting the game...", end="\n\n")

    # Defining printing of the object.
    def __str__(self):

        return f"""Dices color is {self.color}, the sideup is {self.side_up}
and sum of throws is {self.sum_of_throws} \n """

# Setting my_dice as a Dice object
my_dice = Dice()
print("options are = roll the dice / get side up / restart game")

# Infinite loop for the game. user input defines what method is used.
while True:

    user_input = str(input(": "))

    if user_input == "roll the dice":
        my_dice.roll_the_dice()

    elif user_input == "get side up":
        my_dice.get_side_up()

    elif user_input == "restart game":
        my_dice.restart_game()

    else:
        print("Value error.")
```

Screen capture of the output of Task 5



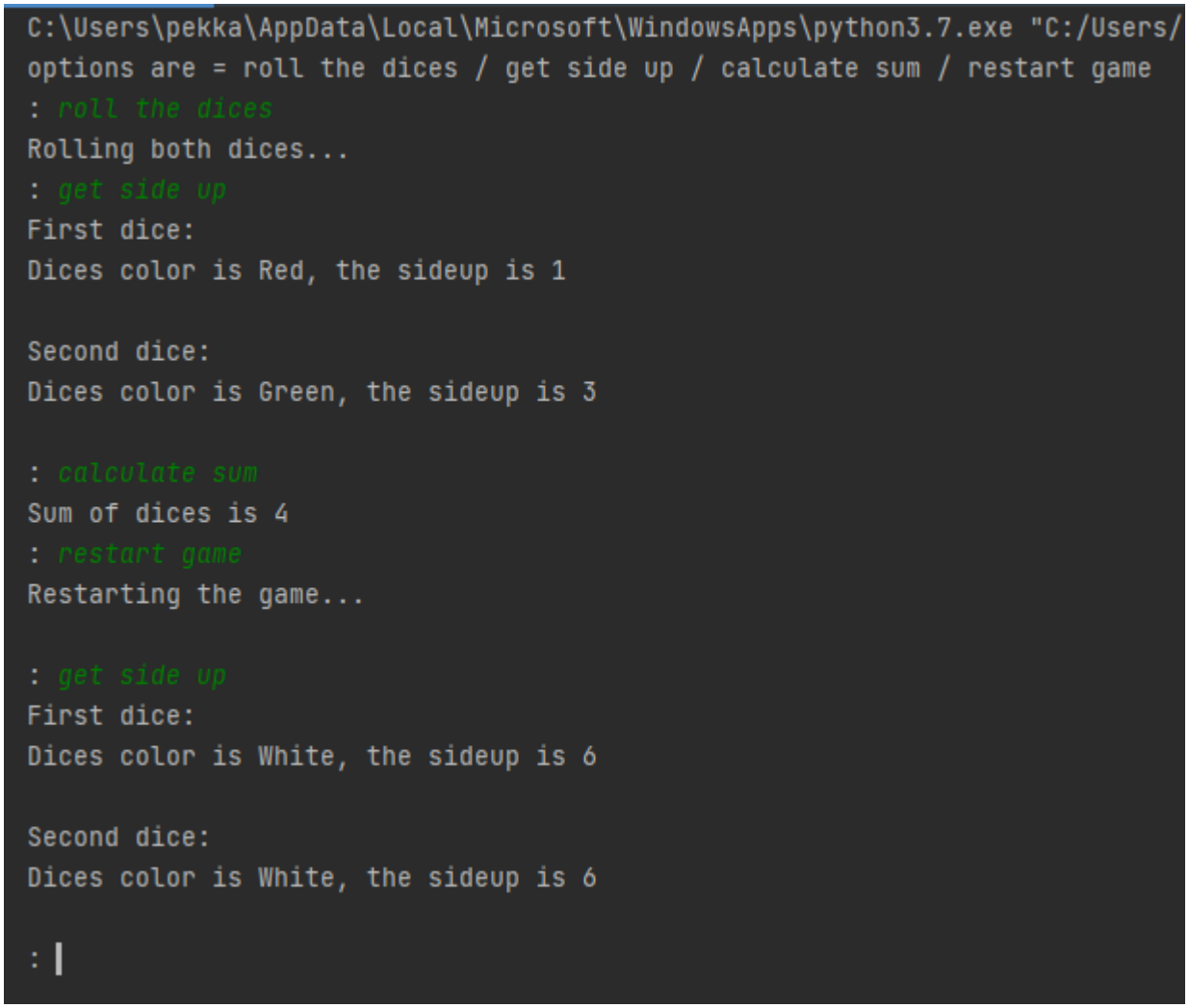
6. Create two Dice objects and roll them both. Sum the result and print to screen.

Screen capture

of Task 6

```python
# File name: Exercise3_6
# Author: Pekka Lehtola
# Description: Roll two dices. can calculate sum of them.

import random

class Dice:

    # Dice with three attributes: Color,
    def __init__(self):

        self.color = "White"
        self.side_up = 6


    # Selecting random number between 1 and 6, form that selectin predefined side color.
    # Modifying dices values and calculating new sum.
    def roll_the_dice(self):

        random_number = random.randint(1,6)
        colors = {1: "Red", 2: "Blue", 3: "Green", 4: "Yellow", 5: "Black", 6: "White"}

        self.side_up = random_number
        self.color = colors[self.side_up]

    #Calculates sum of dices.
    def calculate_sum(self, second):

        sum = self.side_up + second.side_up
        print ("Sum of dices is", sum)

    # Prints objects str method.
    def get_side_up(self):

        print(self)

    # Runs init method again to reset the sum of throws
    def restart_game(self):

        self.__init__()
        print("Restarting the game...", end="\n\n")

    # Defining printing of the object.
    def __str__(self):

        return f"""Dices color is {self.color}, the sideup is {self.side_up} \n """

# Setting first_dice and second_dice as Dice objects
first_dice = Dice()
second_dice = Dice()

print("options are = roll the dices / get side up / calculate sum / restart game")

# Infinite loop for the game. user input defines what method is used.
while True:

    user_input = str(input(": "))

    if user_input == "roll the dices":
        print("Rolling both dices...")
        first_dice.roll_the_dice()
        second_dice.roll_the_dice()

    elif user_input == "get side up":
        print("First dice: ")
        first_dice.get_side_up()
        print("Second dice: ")
        second_dice.get_side_up()

    elif user_input == "calculate sum":

        first_dice.calculate_sum(second_dice)

    elif user_input == "restart game":
        first_dice.restart_game()
        second_dice.restart_game()

    else:
        print("Value error.")

```

Screen capture of the output of Task 6



7. Design first using pseudocode, then code this: Create a Dice rolling game of three players (three Dice objects). On first round everybody rolls their dice, lowest number loses and is out of game. On second round the two remaining contestants roll a dice and higher number wins. Use proper output prints of the situation all the time. If on either round there is a tie between 2 or 3 dices, then the tied dices are rolled again.

Program start

Initialize Dice:

    Side up = 1

    Name = user input

Defining dice roll:

    if player side up = 7:

        Skip

    else:

        Side up = random number between 1 – 6

Defining check duplicates:

    if all players have the same side_up number:

        player_1 dice roll

        player_2 dice roll

        player_3 dice roll

    else If player_1 and player_2 has the same side_up number but not 7:

        player_1 dice roll

        player_2 dice roll

    else If player_1 and player_3 has the same side_up number but not 7:

        player_1 dice roll

        player_3 dice roll

    else If player_2 and player_3 has the same side_up number but not 7:

        player_2 dice roll

        player_3 dice roll

Defining check winner:

create list containing players dice_up numbers

remove from list players that has dice_up value of 7

If list lenght = 1:

Declare remaining player as winner.

Exit code

else:

select the player with the smalles dice_up value from the list

set dice_up value as 7

Defining main:

Infinite loop:

All players roll the dice

check for duplicates

check winner

Screen capture of Task 7

```python
# File name: Exercise3_7_v2
# Author: Pekka Lehtola
# Description: Three player dice game

import random

#Dice class with player name and side up attribute.
class Dice:

    def __init__(self, name):

        self.name = name
        self.side_up = 1

    #If player allready lost they dont roll the dice
    def roll_the_dice(self):

        if self.side_up == 7:
            pass
        else:
            self.side_up = random.randint(1, 6)
            print(self.name, "rolls the dice...and gets", self.side_up)

#Setup players name attribute
player_1 = Dice(input("Name of player one: "))
player_2 = Dice(input("Name of player two: "))
player_3 = Dice(input("Name of player three: "))


#Checks if there are multiple of the same numbers that are not 7
def check_duplicates():

    if player_1.side_up == player_2.side_up and player_1.side_up != 7:

        print(player_1.name, "and", player_2.name, "rolls again")
        player_1.roll_the_dice()
        player_2.roll_the_dice()

    elif player_1.side_up == player_3.side_up and player_3.side_up != 7:

        print(player_1.name, "and", player_3.name, "rolls again")
        player_1.roll_the_dice()
        player_3.roll_the_dice()

    elif player_3.side_up == player_2.side_up and player_2.side_up != 7:

        print(player_3.name, "and", player_2.name, "rolls again")
        player_3.roll_the_dice()
        player_2.roll_the_dice()

    #If every player rolls the same number all players roll again.
    elif player_1.side_up and player_2.side_up == player_3.side_up:
        print("all players need to roll again: ")
        player_1.roll_the_dice()
        player_2.roll_the_dice()
        player_3.roll_the_dice()

#Check_list contains players dice rolls
#Check_list_dict is used for connecting players dice roll with players name
def check_winner():

    check_list = [player_1.side_up, player_2.side_up, player_3.side_up]
    check_list_without_removed_players = []
    check_list_dict = {player_1.side_up: player_1.name, player_2.side_up: player_2.name, player_3.side_up: player_3.name}
```

```
    #Fills check_list without removed_players with players with out dice side_up 7
    for i in range(len(check_list)):
        if int(check_list[i]) != 7:
            check_list_without_removed_players.append(check_list[i])

    #If two other player has side_up 7 declare remaining player as winner.
    if len(check_list_without_removed_players) == 1:
        winner = check_list_without_removed_players[0]

        print("Winner of the game is", check_list_dict[winner])
        exit()

    #Removes player with the smallest number by setting its side_up value as 7
    else:
        smallest_number = min(check_list_without_removed_players)
        player_name = check_list_dict[smallest_number]

        if player_name == player_1.name:
            player_1.side_up = 7
            print(player_1.name, "Had the smallest number and was removed from the game")

        if player_name == player_2.name:
            player_2.side_up = 7
            print(player_2.name, "Had the smallest number and was removed from the game")

        if player_name == player_3.name:
            player_3.side_up = 7
            print(player_3.name, "Had the smallest number and was removed from the game")


def main():

    player_1.roll_the_dice()
    player_2.roll_the_dice()
    player_3.roll_the_dice()

    check_duplicates()
    check_winner()

    player_1.roll_the_dice()
    player_2.roll_the_dice()
    player_3.roll_the_dice()

    check_duplicates()
    check_winner()
    check_winner()

main()
```

Screen capture of the output of Task 7

```
C:\Users\pekka\AppData\Local\Microsoft\WindowsApps\python3.7.exe
Name of player one: Pekka
Name of player two: Matti
Name of player three: Juhani
Pekka rolls the dice...and gets 1
Matti rolls the dice...and gets 3
Juhani rolls the dice...and gets 3
Juhani and Matti rolls again
Juhani rolls the dice...and gets 3
Matti rolls the dice...and gets 2
Pekka Had the smallest number and was removed from the game
Matti rolls the dice...and gets 6
Juhani rolls the dice...and gets 6
Juhani and Matti rolls again
Juhani rolls the dice...and gets 6
Matti rolls the dice...and gets 4
Matti Had the smallest number and was removed from the game
Winner of the game is Juhani

Process finished with exit code 0
```

Design first using pseudocode, then code this: Create a CellPhone Class. Write a program that will design a class that represents a cell phone. The data attributes are manufact (Manufacter), model (Model) and retailPrice (Retail price). The class will also have the following methods:

a. __init__

b. set Manufact

c. set Model

d. setRetailPrice

e. getManufact

f. getModel

g. getRetailPrice

Program start

Defining Cellphone:

Defining initializing:

manufacturer = " "

model = " "

retail price = 0

Defining set manufact:

Cellphone manufacturer = user input

Defining set model:

Cellphone model = user input

Defining set retail price:

Cellphone retail price = user input

Defining get manufacturer:

return "Manufacturer: " + Cellphone manufacturer

Defining get model:

return “Model number: “ + Cellphone model number

Defining get retail price:

return “Retail price: “ + Cellphone retail price

Defining main:

my cellphone = Cellphone

my cellphone set manufact

my cellphone set model

my cellphone set retail price

print “ Here is the data that you provided : “

my cellphone get manufacturer

my cellphone get model:

my cellphone get retail price:

Start main

Screen capture of Task 8

```python
# File name: Exercise3_8
# Author: Pekka Lehtola
# Description: Creates cellphone class from user inputs

class Cellphone:

    def __init__(self):

        self.manufact = ""
        self.model = ""
        self.retail_price = 0

    def set_manufact(self):

        self.manufact = str(input("Enter the manufacturer : "))

    def set_model(self):

        self.model = str(input("Enter the model number : "))

    def set_retail_price(self):

        self.retail_price = float(input("Enter the retail price : "))

    def get_manufact(self):

            return print("Manufacturer:", self.manufact)

    def get_model_number(self):
        return print("Model number:", self.model)

    def get_retail_price(self):
        return print("Retail price:", self.retail_price)

def main():

    my_cellphone = Cellphone()

    my_cellphone.set_manufact()
    my_cellphone.set_model()
    my_cellphone.set_retail_price()

    print("Here is the data that you provided :")

    my_cellphone.get_manufact()
    my_cellphone.get_model_number()
    my_cellphone.get_retail_price()

main()
```

Screen capture of the output of Task 8

```
C:\Users\pekka\AppData\Local\Microsoft\
Enter the manufacturer : Apple
Enter the model number : iPhone7
Enter the retail price : 500
Here is the data that you provided :
Manufacturer: Apple
Model number: iPhone7
Retail price: 500.0

Process finished with exit code 0
```

9. Take a look at the CellPhone Class/Object: where are these concepts (or are they there) (take a screen capture and indicate a line)?

a. Object?

b. Encapsulation?

c. Data attributes?

d. Hidden attributes?

e. Public methods?

f. Private methods?

g. Init-method?

```
# File name: Exercise3_8
# Author: Pekka Lehtola
# Description: Creates cellphone class from user inputs

class Cellphone:

    def __init__(self):  <- Init-method

        self.manufact = ""
        self.model = ""
        self.retail_price = 0   <- Data attributes

    def set_manufact(self):

        self.manufact = str(input("Enter the manufacturer : "))

    def set_model(self):

        self.model = str(input("Enter the model number : "))

    def set_retail_price(self):

        self.retail_price = float(input("Enter the retail price : "))

    def get_manufact(self):

            return print("Manufacturer:", self.manufact)

    def get_model_number(self):
        return print("Model number:", self.model)

    def get_retail_price(self):
        return print("Retail price:", self.retail_price)

def main():

    my_cellphone = Cellphone()  <- Object

    my_cellphone.set_manufact()
    my_cellphone.set_model()
    my_cellphone.set_retail_price()

    print("Here is the data that you provided :")

    my_cellphone.get_manufact()
    my_cellphone.get_model_number()
    my_cellphone.get_retail_price()

main()
```

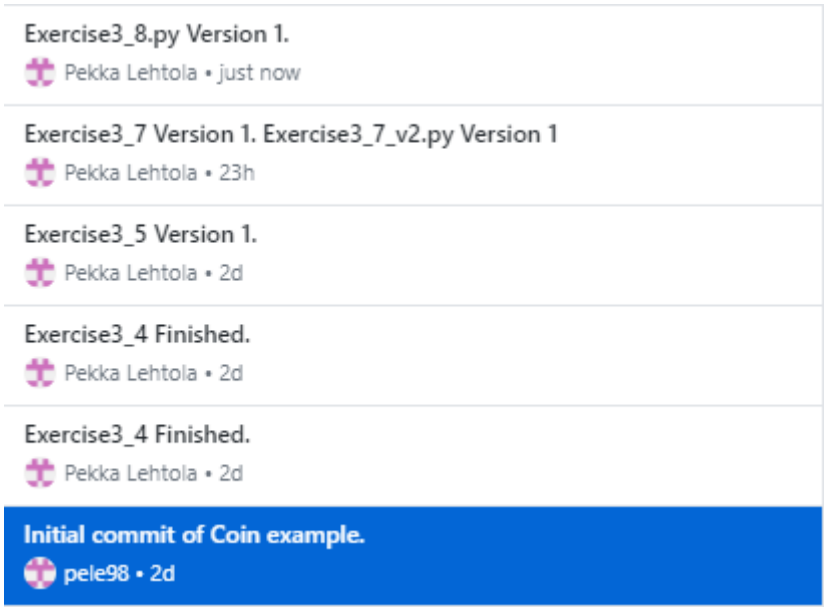
<- Public methods

No Encapsulation in this code, that would require private data attributes like _model or __manufact

No Hidden attributes

No private methods

Screen capture of git log (showing that you made a commit after every task).



Huom. Tehtävät 2-4 samassa koska oli teknisiä ongelmia…

Self-assessment:

This exercise was easy/difficult/ok/etc. for me because…

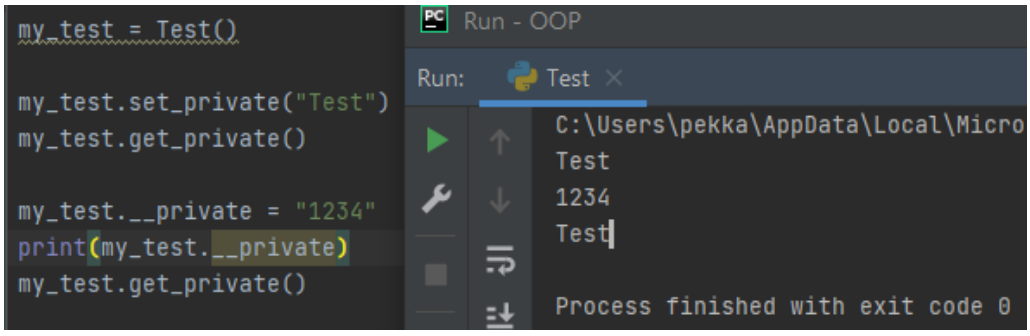
Näiden tehtävien kanssa jostain syystä koin todellisia tuskia, varsinkin tehtävässä 7, jotenkin en saanut päätäni toimimaan kun yritin toteuttaa usealla luokalla.

Doing this exercise, I learned…

Edellisissä kotitehtävissä miettisin __str__ metodin tarkoitusta ja näiden tehtävien jälkeen tarkoitus on selkeä.

I am still wondering…

En ihan ymmärrä mitä tässä tapahtuu…



I understood/did not understand that… ; I did/did not know that… ; I did/did not manage to do…

Tosiaan subclassit jäi laajankin kokeilujen jälkeen mysteriksi.